Schedule

Week	Tuesday	Thursday
Week 1 (09/02-09/06)	1, 2	3
Week 2 (09/09-09/13)	4.1-4.3	4.4-4.5
Week 3 (09/16-09/20)	5.1	5.1
Week 4 (09/23-09/27)	5.2	5.3-5.7, Coding
Week 5 (09/30-10/04)	Midterm 1[1]	6.1, 6.2
Week 6 (10/07-10/11)	7	7
Week 7 (10/14-10/18)	Break	7
Week 8 (10/21-10/25)	8	8
Week 9 (10/28-11/01)	9	9
Week 10 (11/04-11/08)	9	Midterm 2[2]
Week 11 (11/11-11/15)	CFGs	CFGs
Week 12 (11/18-11/22)	CFGs	PDAs
Week 13 (11/23-11/29)	PDAs	Thanksgiving
Week 14 (12/02-12/06)	Review	Review

Week 1 (09/03-9/07)

Day 1

Goals
- Understand the different kinds of problems we would investigate, with Python examples.
- Understand what "Python programs" we will consider in this class.
- Practice with string representations of standard kinds of inputs.

Chapter 1
- Introduction to Theory of Computation
- Different kinds of problems

Activity 1
- Discuss examples of tractable/intractable/uncomputable problems.
- Activity Handout[3]

Chapter 2
- What is a SISO Python program?
- Input and Output of a program.
- Practice programs (pseudocode sufficient, work at the board):
  - Takes a string input and runs forever (at least on some inputs).
  - Takes a string input and throws an exception (at least on some inputs).
  - Given a string representing a list of numbers separated by spaces, like "23 45 13", add the numbers up and return the sum as a string.
- Exercise 2.5 a-d (work at the computers).

[1] midterm1_study_guide.html
[2] midterm2_study_guide.html
[3] activities/activity1.html

- When are SISO programs equivalent? Examples of different but equivalent programs.
- Decision programs.
- Exercise 2.7 (talk with neighbor)

Activity 2
- How can a SISO program process the following kinds of inputs:
  - A list of an arbitrary number of integers.
  - Two strings of predetermined length (say each of length 20).
  - Two strings of non-predetermined length.
  - Three strings of non-predetermined length.
  - A list of an arbitrary number of strings of non-predetermined length.

Activity 2b Write a SISO program that will take as input a string that represents two integers (based on some convention you establish for how to do that) and will return the string that represents the sum of those integers.

Day 2

Goals
- Understand the fact that programs are simply strings and can be inputs to other programs.
- Understand the difference between running a program and reasoning about a program's execution.
- Understand the limitations of programming languages by showing there are programs that cannot exist.

Chapter 3
- Examples of decision programs: containsGAGA, yes, longerThan1K, maybeLoop.
- The countLines program; programs as input to other programs.
- Self-reflection: programs examining themselves (group work Figure 3.3).
- The program yesOnString. Try at various inputs (group work Figure 3.4).
- Python offers a command exec, which takes a Python file and executes it. Can we use it to implement the behavior described by yesOnString (group discussion)?
- The program yesOnSelf. (group work Figure 3.5).
- Write yesOnSelf using yesOnString as a helper.
- The program notYesOnSelf. Think through what happens when notYesOnSelf is run with input the earlier programs, as well as itself. (group work).
- Discussion of programs crashOnString, crashOnSelf, weirdCrashOnSelf.

Activity 3
- Discuss definitions of the outputs of the programs noOnString, noOnSelf.
- Are noOnSelf and notYesOnSelf equivalent programs, if they existed?
- Is a similar conclusion possible for noOnSelf as for notYesOnSelf?
- Exercise 3.3

- Activity Handout[4]

HW 1 (Due 9/12 4pm in LYN 110) 3.2, 3.10, 3.11

Week 2 (09/10-09/14)

Day 1

Goals
- Review the basic methodology of proof by contradiction.
- Understand the terms Alphabet, Language, String, in the Theory of Computation context.

Chapter 3 recap
- Exercise 3.3
- Review of proofs by contradiction.
- Write a formal proof that the program yesOnString cannot exist.

Section 4.1
- Computational Problems and how they differ from programs
- Alphabets, Strings, (Formal) Languages
- Exercise 4.3 (group discussion)
- Various examples of languages and language constructions
  - Empty/All, containing empty string only, Python/Java programs
  - Strings accepted by a decision program
  - union/intersection/complement/concatenation/Kleene
- Activity 4, first two points

Activity 4
- Describe three formal languages that are sufficiently different from those listed in the book.
- Exercises 4.4, 4.6

Day 2

Goals
- Learn the basic terminology related to computational problems
- Understand what it formally means to solve a problem.

Review
- Consider the alphabet $\Sigma = \{a, b\}$.
  - Give a clear English description of the language Σ^*.
  - How many strings of length exactly 4 are there in Σ^*?
  - List all strings $s \in \Sigma^*$ with $|s| < 3$ in **lexicographic order**.

Sections 4.2-4.4
- Computational problems:
  - inputs, solution sets, positive and negative instances
  - Describe the SHORTESTPATH problem

[4] activities/activity3.html

  - Exercises 4.11, 4.13
  - Kinds of computational problems: search, optimization, threshold, function, decision
  - General computational problems vs decision problems. Figure 4.13
  - Exercise 4.15
- What does it mean to solve/compute/decide a computational problem? Provide both positive and negative examples.
- Computable vs uncomputable problems. Provide examples of uncomputable problems.

Practice
- Exercises 4.5, 4.7
- The problem GREATER takes as input a number, and has as solutions all numbers that are greater than this number. Is this a computational problem according to our definition?
- A *composite* number is a positive integer which has at least one non-trivial divisor. The problem FACTOR takes as input a number. If the number is composite, then the solution set is all of the number's non-trivial divisors. If the number is prime, the solution is "no" indicating no solution. This can be described as a "search" problem. Express it as a search problem by specifying a predicate function.
- Define "threshold", "optimization" and "decision" versions of the previous problem that have an analogous computational difficulty.
- Write a Python program that *decides* the language that contains all strings in the ASCII alphabet that have "matching parentheses", meaning that every "open parenthesis" is going to be matched by a "close parenthesis". The strings can contain any other letters inbetween.
- (Optional, Challenge) Using HasShortPath as a helper function, write an algorithm for solving ShortestPath.

HW 2 (Due 9/19 4pm in LYN 110) 4.12, 4.19, 4.20, 4.25a,b, Bonus problems: 4.25c, 4.26

Week 3 (09/17-09/21)

Day 1

Goals
- Understand the difference between decidable and recognizable languages.

Section 4.5
- The membership problem for a language. The language corresponding to a decision problem.
- Definition of decidable languages and decidable decision problems.
- Is the language of Java Programs decidable?
- Recognizable languages. Give examples of recognizable languages that are not decidable.

Activity 5
- Write a Python program that *decides* the language of all (finite) binary strings that contain at least two 1s. (Food for thought: does this program have to worry about handling non-binary strings?)
- Write a Python program that *decides* the language of all prime numbers.
- Write Python programs that *decide* the empty language and the language consisting of just the empty string, respectively.

Day 2

Goals
- Understand the definition of a Turing machine
- Understand the graphical representation of a Turing machine
- Be able to follow the execution of a Turing machine on a given input

Section 5.1
- Definition of a Turing machine: alphabet, states, transition functions
- Transition function can be thought of as a combination of: new state function, new symbol function, direction function
- What different final states can we have in a Turing machine? Are any of them required?
- Write a Turing machine that given a string input accepts if the input has even length.
- Describing a Turing machine via state diagrams instead of tables of transitions.
- Abbreviated notation for state diagrams.
- Show a complete computation.
- Activity 6, first one
- Write a Turing machine that inserts an x at the beginning of the input, shifting all other letters to the right, and also inserts an x at the end of the input.
- What is the difference between *looping* and *halting*?
- What Turing machines do we call *transducers* and what machines do we call *accepters*? Are those the only kinds of Turing machines?

Activity 6
- Make a state diagram for a Turing machine that given a string input accepts if the input has even number of vowels and rejects otherwise.
- Make a state diagram for a Turing machine that searches for the first C that occurs in the string and changes it to a G, then halts. If it arrives at the end of the input without finding a C, then it inserts a C and halts.
- Consider the alphabet consisting of only the binary digits 0, 1. Consider the input as a number with the least significant digit being at the beginning of the tape. So if the number was 8, the tape would start with "001" with the first 0 being at the start of the tape. We will call this representation of numbers *reverse binary*. Make a state diagram for a Turing machine that would change the tape contents so that the final output is representing the input number incremented by 1. So in the example above the output would be "101". And if we had started with "101" as input, then the output would be "011" etc.

Homework 3 (Due 9/28 4pm in LYN 110)
- 4.25b, 5.3 (but change it to "at least 2 Gs and at most 3 Ts"), 5.5, 5.6
- Coderunner problem isPrime in Moodle

Week 4 (09/24-09/28)

Day 1

Goals
- Understand how a Turing machine can use the tape to remember an unbounded amount of information.

Section 5.2
- A Turing machine that checks whether the string has more Cs than Gs. It follows two different paths depending on whether it encounters a G or a C first (decider).
- Discuss "reverse binary" notation and operations.
- Implement a Turing Machine that takes a "reverse binary" input enclosed by a pair of xs and increments its value by 1 (transducer).
- Practice: Describe the expected behavior of a reverseBinaryDecrementer machine (should reject if the input is the number 0) and construct its state diagram.

Day 2

Goals
- Understand how a Turing machine can be used as a component/subprogram in another Turing machine.
- Understand how two-tape and multi-tape machines have the same power as one-tape machines.
- Understand how a Python program can simulate a Turing machine.
- Understand how Python programs are computationally equivalent to Turing machines.
- Practice writing and running Turing Machines in Python using simulateTM.

Section 5.2
- Go through Problem 4.25
- Use the reverse binary incrementer to implement a countCs Turing machine. Start by describing these helpers:
  - prependx
  - prepend0
  - incrementReverseBinary
  - moveHeadToNumberStart
  - moveHeadToStringStart
  - deleteString

Activity 7
- Create a state diagram for a Turing machine that compares two numbers as follows:

  - The numbers are provided in binary form from highest significant number to lowest, separated by x's and with zeroes padded in if needed to make sure both numbers have the same number of digits. For example the numbers 6 and 2 would be represented as: "x110x010x".
  - Your machine is free to change the tape contents as it needs to. You may also introduce up to two new symbols: y and z.
  - Your machine should accept if the first number is at least as large as the second, and it should simply halt if the second number is larger.
  - Is this machine a decider for a language? Explain.

Sections 5.3-5.7
- A two-tape single-head Turing machine can be simulated by a standard Turing machine by using an alphabet consisting of pairs of characters from the alphabet of the original machine (a "squared alphabet").
- A multi-tape single-head Turing machine can be simulated in a similar way.
- A multi-tape multi-head Turing machine can be simulated by using a multi-tape single-head Turing machine that uses twice the number of tapes, and uses the second set of tapes to keep track of the locations of the multiple heads.
- How can a Python program be used to simulate a Turing machine?
- Talk about how it's plausible that a Turing machine has the same computational power as Python.

Activity 8 Activity 8 writeup[5]

Homework 4 (Due 10/05 4pm in LYN 110)
- 5.7, 5.13. For each Turing machine you create provide both a state diagram and a text description of the Turing machine as in the provided .tm files. (See also Fig. 5.19)
- Write and test a single-tape Turing machine that implements the "**reverse** binary decrementer" that we discussed in class. Submit the printout of your .tm file. Include a state diagram.

Week 5 (10/01-10/05)

Day 1

- Midterm 1[6]

Day 2

Goals
- Understand the idea of *universal computer programs* and *universal Turing machines*.
- Familiarize yourself with programs that alter other programs.
- Use a universal computer program to produce programs that are recognizers but not deciders.

[5] activities/activity8.html
[6] midterm1_study_guide.html

Sections 6.1, 6.4, 6.5 (6.3 optional)
- Understand the program universal.py and how it works.
- Describe the "simulate-and-alter" technique for changing a program's behavior (Figure 6.8).
- Group activity: 6.4
- Describe the input-ignoring program in Figure 6.9.
- Show that YesOnString is recognizable.
- Group activity: 6.6 (Actually write the program and run it on positive and negative instances)

Activity 9
- Construct a program that demonstrates that the language CrashOnString is recognizable.

Homework 5 (Due 10/11 4pm in LYN 110) 6.3, 6.7, 6.9 (make sure to actually write a Python program for it)

Week 6 (10/08-10/12)

Day 1

Goals
- Understand what it means to have a *reduction* from problem A to problem B.
- Use reduction arguments to prove one problem is *at least as hard* than another.

Sections 7.1, 7.2, 7.3
- The IsOdd and LastDigitIsEven problems, and solving IsOdd via LastDigitIsEven.
- Using a program that solves one problem to build a program that solves another.
- Showing a program is impossible by using it to write a program that is known to be impossible.
- Informal definition of the phrase "problem F reduces to problem G"
- The problem GAGAOnString and proving it is undecidable via a reduction from YesOnString:
  - Transforming *instances* of YesOnString to instances of GAGAOnString.
  - Positive instances must map to positive instances, and negative instances to negative instances.
  - How to handle "ill-formed" negative instances.
  - Writing the program altersYesToGAGA.
  - Tying it all together.
- Talk about a reduction from yesOnString to containsGAGAOnString.
- Activity 10, first part.
- Activity 10, second part.

Activity 10
- Consider the decision problem IntOnString: It takes two strings P, I as input, and outputs "yes" if P is a program and $P(I)$ is a non-negative integer string (i.e. containing only digits). Similarly to what we did for GAGAOnString, show how YesOnString can be reduced to IntOnString.
- Suppose F, G are general computational problems, and D is a decision problem. Also suppose that F reduces to G, and also D reduces to G. What, if anything, can we conclude if:
  1. D is undecidable
  2. G is computable
  3. F is computable

Day 2

Sections 7.4, 7.5
- Definition of Turing reduction.
- What is an Oracle program.
- Propagation of computability.
- Is the converse propagation true?
- Turing reducibility is transitive.
- The problems yesOnEmpty, yesOnAll, yesOnSome.
- Refresher on the program ignoreInput.
- Use ignoreInput.py to write yesOnString using yesOnEmpty as a helper.

Homework 6 (Due 10/24 11am in LYN 110) 7.3, 7.5b,c,e (using the "explicit Python program" technique 2 from p. 138), 7.7

Week 7 (10/15-10/19)

Day 1

- Fall Break

Day 2

Section 7.5
- Review of reduction from yesOnString to yesOnEmpty or yesOnSome or yesOnAll. Sequence diagrams visual.
- Definition of what it means for a Python program to *halt*.
- The Halting problem and its variants: haltsOnString, haltsOnEmpty, haltsOnAll, haltsOnSome.
- Use an altering technique similar to what we did for GAGAOnString to show a reduction from yesOnString to haltsOnString.
- Group activity: Show reduction from haltsOnString to haltsOnEmpty. Include code and sequence diagram.

Section 7.6
- Group activity: Figure 7.16. Which of these programs actually compute IsEven? Explain.
- The problem Computes_F for a computational problem F.
- If F is a computable problem, then Computes_F is in fact uncomputable.
- The problem ComputesOneOf_S.

Week 8 (10/22-10/26)

Day 1

Sections 8.1, 8.2, 8.3
- Parallel computation in Python programs
- A parallel computation Python program for containsNANA
- Considering this program as a non-deterministic program
- Computation Trees, positive/negative/non-terminating leaves
- Definition of the output of a nondeterministic computation
- Writing pseudocode for recognizer for the concatenation of two recognizable languages.
  - Conclusion: The concatenation of recognizable languages is recognizable.

Day 2

Leftover
- "Truth table" for "threaded or"
- Group activity: "threaded and"
- Exercise 8.11: The union of recognizable languages is recognizable.

Sections 8.4, 8.5
- Non-deterministic Turing Machines
- Example of a NDTM detecting if a genetic string contains AAT
- Example of a computation in a NDTM: The string AAGAT
- Group activity: Describe the full nondeterministic computation for this problem and the string GAATG
- Recognizing the union of two recognizable languages via NDTMs

Sections 8.6, 8.7, 8.8
- Formal definition of NDTM
- For every NDTM that solves a problem, there is a DTM that solves it.
- If a language and its complement are both recognizable, then the language is decidable.

Homework 7 (Due 10/30 4pm in LYN 110) 8.4, 8.7, 8.8, 8.9

Week 9 (10/29-11/02)

Day 1

Homework review
- Exercise 7.7

Chapter 8 wrap-up
- Let L be an undecidable language. If L is recognizable, then the complement of L is unrecognizable.
- Co-recognizable languages. Venn diagram of languages and recognizability.
- Examples of unrecognizable languages.

Sections 9.1, 9.2
- Definition of a Deterministic FA (DFA)
- Finite Automata and their difference from Turing Machines
  - Only read the tape
  - Blank symbol transitions to accept/reject state
  - No halt state
  - Possible results of running a DFA on a string input
- Notation for finite automata
- The "standard" definition for DFAs
- Non-deterministic FAs (NFAs)
  - Allowing epsilon transitions
- NFA accepting a string
- Example on figure 9.5
- Group activity: Draw an NFA that accepts exactly the strings that are concatenation of zero or more of the strings aab or ac (in any order). For example: aabacacaab would be accepted. Use as few states as possible (doable with 5).

Section 9.3
- Every nfa is equivalent to some dfa
- Standard proof: The dfa states represent sets of states from the nfa
  - Figure 9.6
  - Epsilon closures
- Group activity: Construct a dfa equivalent to the nfa in problem 9.7
- Group activity: Construct a dfa equivalent to the nfa you built for strings of aab and ac earlier.

Homework 8 (Due 11/06 4pm in LYN 110): 9.3, 9.7a, 9.8, 9.9

Day 2

Section 9.3 Wrap-up
- Strict nfas
- Converting from an nfa to a strict nfa (Figure 9.5)

Section 9.4
- Pure regular expressions

  - Examples on page 177
- The language represented by a regular expression
- Standard regular expressions
  - Dot, plus, square brackets
- Converting from a regular expression to an NFA recognizing the same language
  - Single letters
  - The empty string
  - Concatenation of two regexs
  - Alternative of two regexs
  - Kleene star of two regexs
- Converting a dfa to a regex:
  - Reduce the dfa, labeling transitions with regexs instead of just alphabet letters
  - At each stage, eliminate a non-start non-accept state and update transitions
  - Result is a dfa like bottom of page 179

Week 10 (11/05-11/09)

Day 1

Sections 9.5-9.7

- Regular languages
- Combining regular languages:
  - The union, complement, intersection, star, and reverse of regular languages is regular
- The non-regular language GnTn.
  - Intuitive explanation of why this language is not regular
- Idea of pumping before n.

Day 2

- Midterm 2[7]

Week 11 (11/12-11/16)

Day 1

Sections 9.6-9.7 wrap-up

- Proving a language is not regular (Pumping Lemma)
- Examples of non-regular languages

[7] midterm2_study_guide.html

- Language of properly-nested parentheses

CFG notes 5.1
- Motivation for CFGs: Interpreting programming language code
  - Challenges when processing an arithmetic expression
- Informal CFG (recursive) description of the properly-nested-parentheses language.
  - Show two different grammars

Homework 9 (Due 11/20 4pm in LYN 110): 9.12, 9.13, 9.17a,b,f,g

Day 2

CFG notes 5.1, 5.2 (5.2.1-5.2.3)
- Informal description of grammar for GnTn and matched parentheses
- Definition of CFGs (and CFLs):
  - Terminals/(Terminal) symbols
  - Non-terminals/variables
  - Start variable
  - Production rules
- A grammar for (tokenized) arithmetic expressions involving a token/terminal v representing variables, a token/terminal n representing numbers, and tokens/terminals for the operations of addition ($+$) and multiplication ($*$).
- Group activity: Write down a grammar for palindrome strings on the symbols a, b, c.
- Group activity: Expand the grammar for arithmetic expressions to also include parentheses.
- Show that every regular language is context-free.
- Building strings in a grammar:
  - recursive inference (body to head)
  - derivation (head to body)
  - Examples from the matched parentheses language
- The Parse tree corresponding to a derivation
  - Labeling of nodes and leaves
  - Examples of derivations in our arithmetic expressions language.

Homework 10 (Due 11/28 4pm) CFG Notes (page 168) 5.1.1a,b,c, 5.1.2, 5.2.1

Week 12 (11/19-11/23)

Day 1

CFG wrap-up
- Notation for derivations

- Sentential forms
- Origin of name "context-free"
- Group activity: For the following grammar and list of strings, produce derivations of those strings from the grammar, as well as corresponding parse trees:

  ```
  S –> T | E + T
  T –> F | T * F
  F –> v | n
  ```

  Strings:

  ```
  v * n
  v + v + n
  n * v + v
  ```

Pushdown Automata (Notes[8])

- Mention pumping lemma for CFLs and examples of non-CFLs
- Pushdown automata
  - definition
  - basic example
  - emptying the stack of a pda

Homework 11 (Due Tue 12/4 4pm in LYN110)

1. Work out a PDA for the language that consists of all palindromes on the alphabet $\{a, b, c\}$. Show the computation for accepting the string abcba.
2. Work out a PDA for the language that consists of all strings of matched parentheses (e.g. (()(()))).
3. Write the PDA for the grammar on the alphabet { i, e } with rules S–> epsilon | SS | iS | iSe. Follow the standard algorithm for determining the PDA of a CFG, and not some other way. Write out the complete PDA diagram, including the expanded form of the "loops" corresponding to production rules.

Day 2

PDAs The PDA(s) corresponding to a CFG

Parsing

- Notes[9]
- First and Follow sets
  - Polish notation grammar
  - Group activity: First and Follow sets for Polish notation grammar
- LL parsers
  - LL parser for Polish notation grammar
  - Building an LL(1)-table
  - Conflicts for left-recursive grammars
  - "Fixing" a left-recursive grammar
  - Group activity: LL(1)-table for the fixed left-recursive grammar

[9] notes/parsing.html

Week 13 (11/26-11/30)

Day 1

- Notes[10]
- Parsing
  - LR parsers

Homework 12 (Not to turn in)

We define the language of "L-values" as follows:

- The terminals are `v`,`n`,`[`,`]`.
- An L-value can be a single variable name, indicated by the terminal symbol `v`.
- An L-value can be an expression of the form `L[I]`, where `L` is an arbitrary L-value and `I` is the "index", described below:
- The index `I` can be either an integer (represented by `n`) or another L-value expression.

Here is an example string that this grammar should be able to generate: `v[v[i][v]]`.

a. Write a CFG for this language, using the above descriptions to write the corresponding rules. Use `L` for the start variable, and `I` for the index variable. You will need 4 production rules.
b. Compute the first sets for this grammar.
c. Compute the item-sets for the LR(1) parser for this grammar, and show the rightmost derivation, stack and input progress and parse tree for the string `v[v[i][v]]`.
d. Explain clearly why this grammar cannot have a corresponding unambiguous LL(1) parser.
e. Rewrite the grammar to eliminate the left-recursive rule. You will need one new variable, `X`, and 6 production rules total.
f. Demonstrate the leftmost derivation of the string `v[v[i][v]]` in this grammar.
g. Compute first and follow sets for this new grammar, and construct an LL(1) parser for it.
h. Show the stack and input progress in the LL(1) parser for the string `v[v[i][v]]` and this new grammar.

Day 2

- Thanksgiving Break

[10] notes/parsing.html

Week 14 (12/03-12/07)

Day 1

- TBD

Day 2

- Review
- Final Exam Study Guide[11]

[11] midterm3_study_guide.html